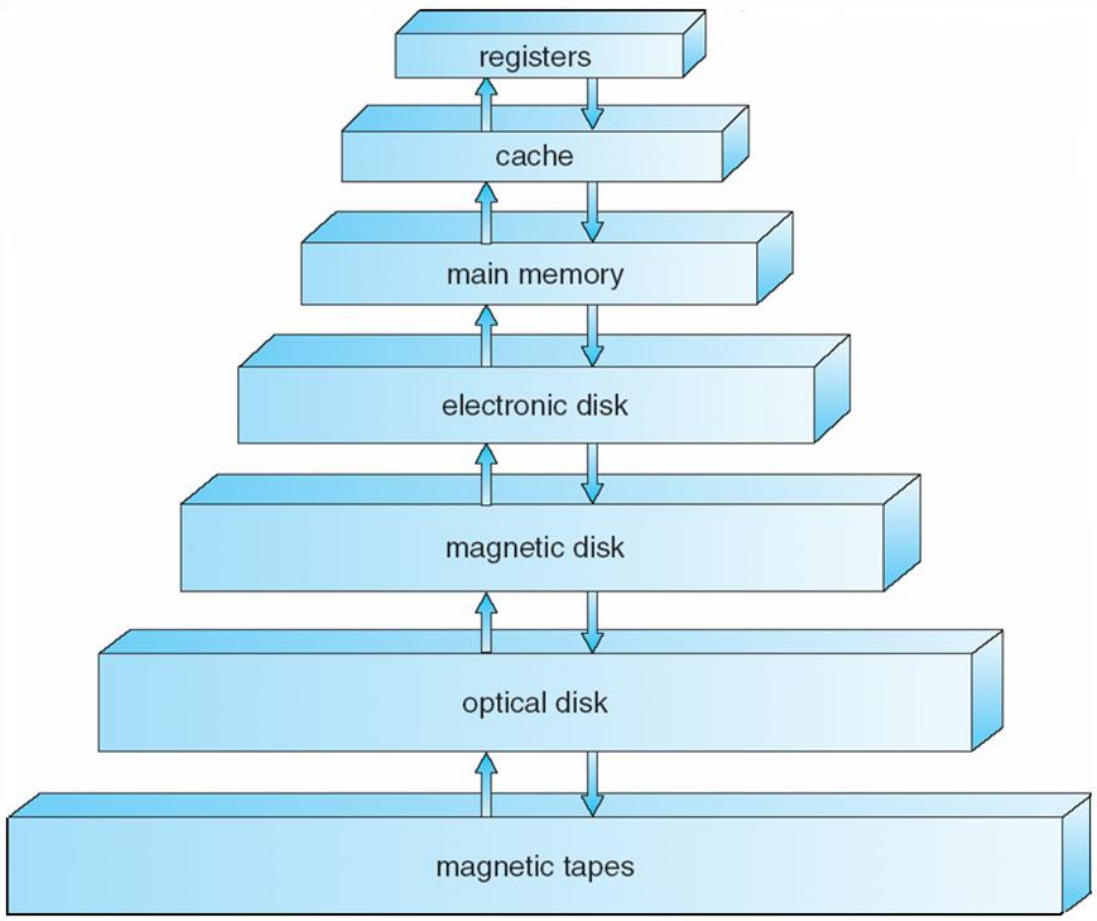
Introduction: virtual memory management

- page management strategies:
  - fetch
  - placement
  - replacement


registers
cache
main memory
electronic disk
magnetic disk
optical disk
magnetic tapes

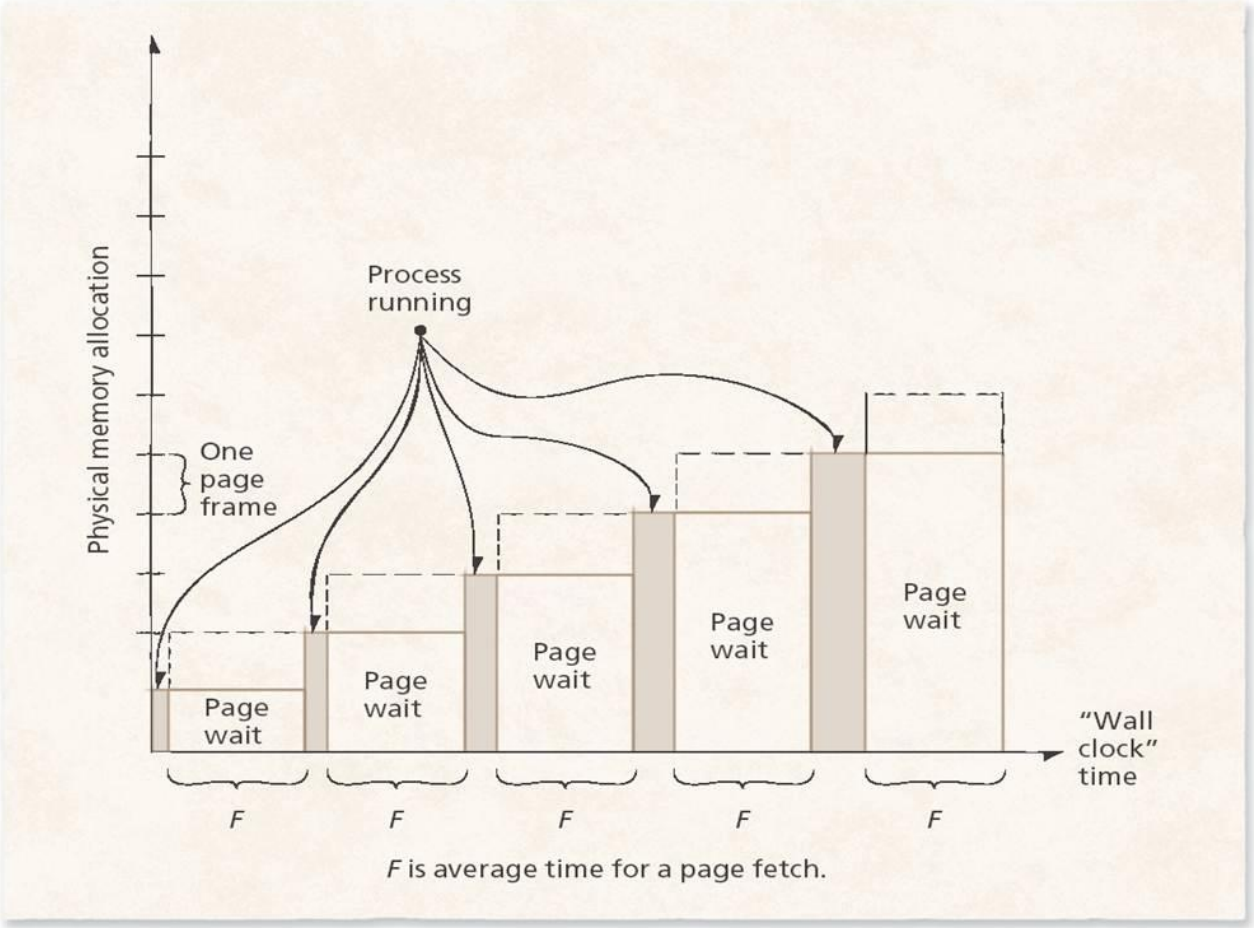
Basic concept: Locality

- Process tends to reference memory in highly localized patterns
  - referenced pages tend to be adjacent to one another in process's virtual address space

Fetch Strategy: Demand Paging

- Demand paging
  - When a process first executes, the system loads into main memory the page that contains its first instruction
  - After that, the system loads a page from secondary storage to main memory only when the process explicitly references that page
  - Requires a process to accumulate pages one at a time

Demand Paging



- waiting process occupies memory

Fetch Strategy: Anticipatory Paging

- attempt to predict the pages a process will need and preloads these pages when memory space is available

- must be carefully designed so that overhead incurred by the strategy does not reduce system performance

Page Replacement

- on page fault:
  - find referenced page in secondary storage
  - load page into page frame
  - update page table entry
- Modified (dirty) bit
  - Set to 1 if page has been modified; 0 otherwise
  - Help systems quickly determine which pages have been modified
- Optimal page replacement strategy (OPT or MIN)
  - Obtains optimal performance, replaces the page that will not be referenced again until furthest into the future

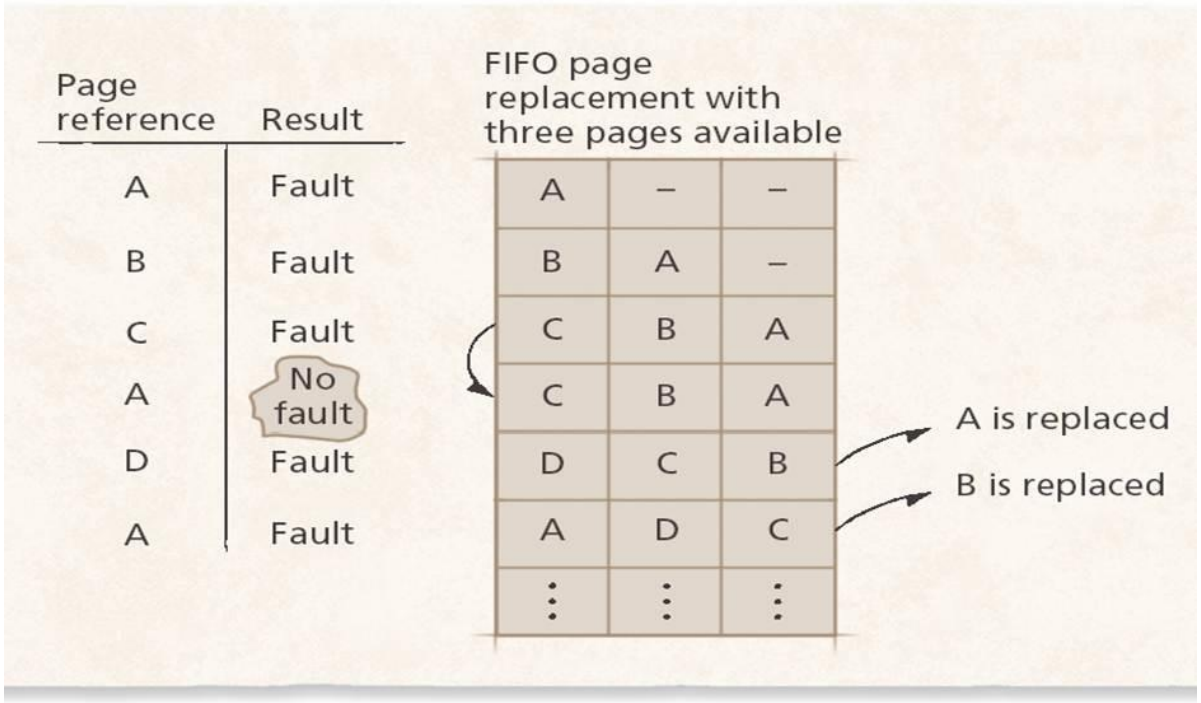
Page Replacement Strategies

- characterized by
  - heuristic it uses to select a page for replacement
  - overhead it incurs
- overview of strategies:
  - FIFO
  - LRU Least-Recently-Used
  - LFU Least-Frequently-Used
  - NUR Not-Used-Recently
  - Second chance & clock page
  - Far page

Random Page Replacement

- low-overhead
- no discrimination against particular processes
- each page has an equal likelihood
- but:
  - could easily select as the next page to replace the page that will be referenced next
  - rarely used

First-In-First-Out (FIFO) Page Replacement



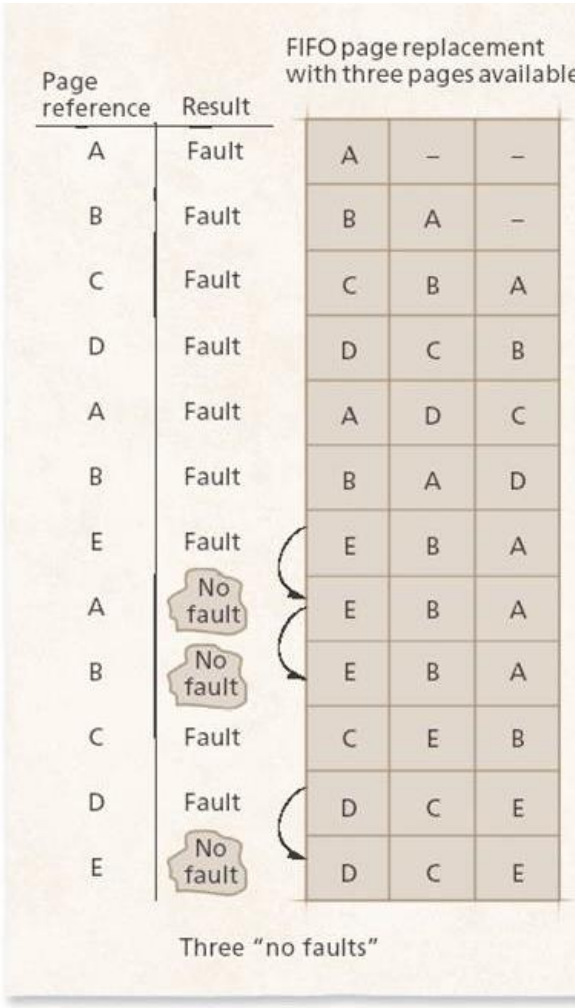
Replace oldest page

- Likely to replace heavily used pages
- relatively low overhead:

  simple queue

- Impractical for most systems

Belady's Anomaly



Least-Recently-Used (LRU) Page Replacement

- Heuristic
  - temporal locality
  - replace page that has not been used recently
- but:
  - increased system overhead
    - list of pages used, update for every page use
  - poor performance in certain situations:
    - large loop

Least-Frequently-Used (LFU) Page Replacement

- Heuristic:
  - keep pages that are being used
  - replaces page that is least intensively referenced
- but:
  - implementation overhead
    - counter for each page ?
  - A page that was referenced heavily in the past may never be referenced again, but will stay in memory while newer, active pages are replaced

Not-Used-Recently (NUR) Page Replacement

- Heuristic:
  - goal: approximate LRU with less overhead
  - uses 2 indicator bits per page:
    - referenced bit
    - modified bit
  - bits are reset periodically
  - order for page replacement
    - un-referenced page
    - un-modified page
- supported in hardware on modern systems

FIFO Variation: Second-Chance Replacement

- Examines referenced bit of the oldest page
  - If off: page is replaced
  - If on:
    - turns off the bit
    - moves the page to tail of FIFO queue
  - keeps active pages in memory

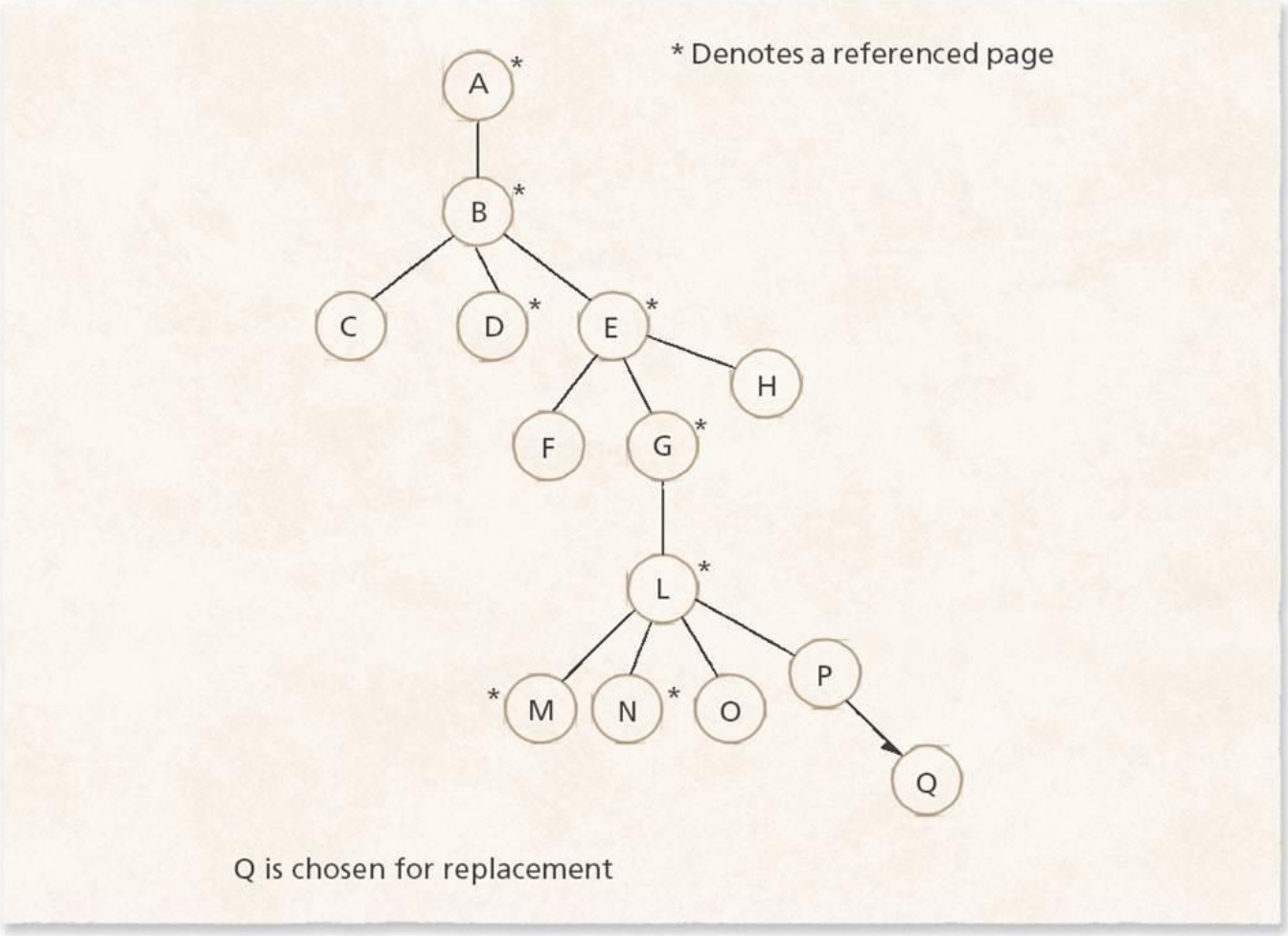
FIFO Variation: Clock Page Replacement

- Heuristic:
  - uses circular list instead of FIFO queue
  - marker for oldest page
- Examines referenced bit of the oldest page
  - If off: page is replaced
  - If on:
    - turns off the bit
    - advances marker in circular list

Far Page Replacement

- Heuristic:
  - Creates an access graph that characterizes a process's reference patterns
  - Replace the unreferenced page that is furthest away from any referenced page in the access graph
  - Performs at near-optimal levels

Far Page Replacement: access graph



Far Page Replacement

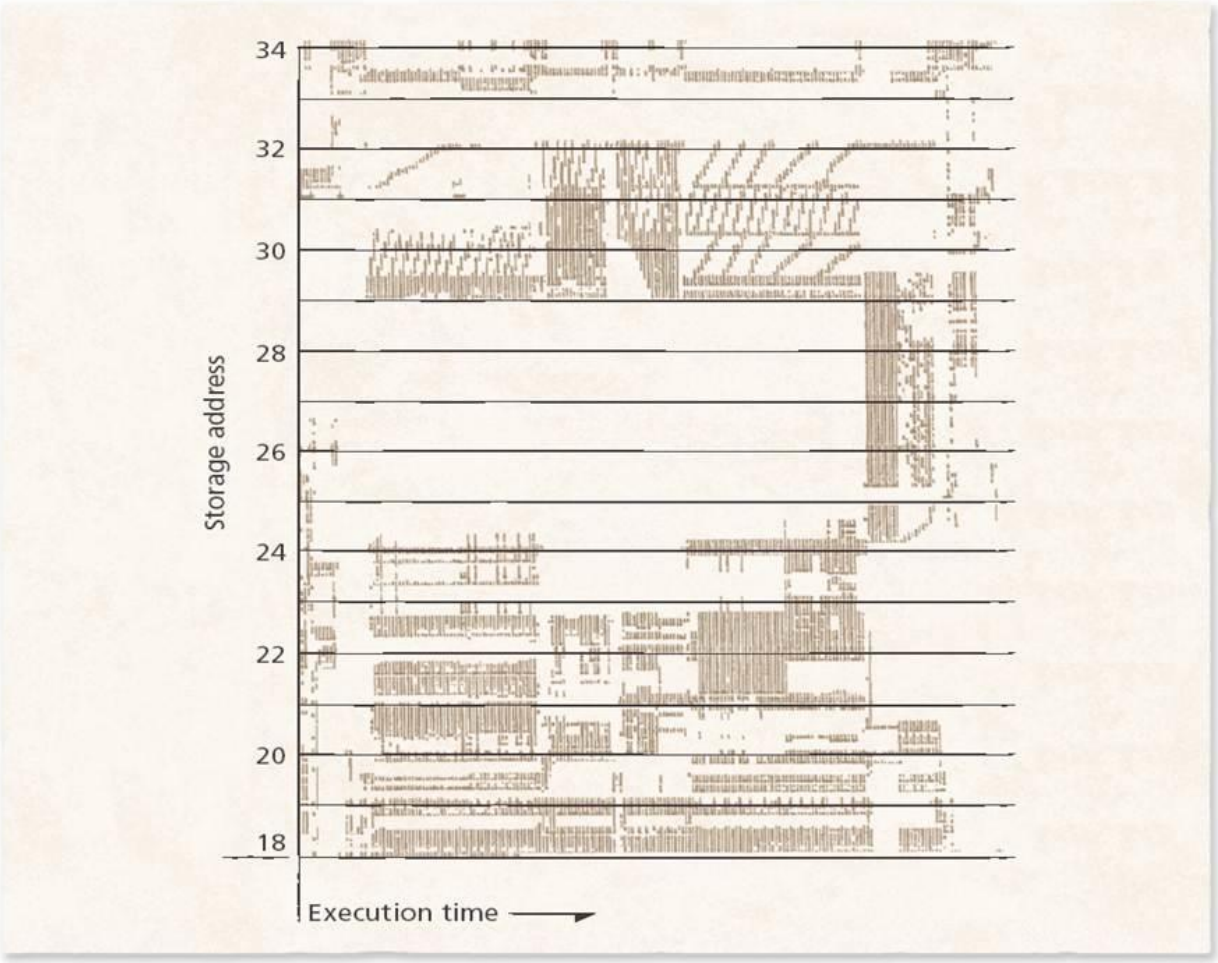
- Performs at near-optimal levels

- but:
  - access graph needs to be computed
  - access graph is complex to search and manage without hardware support

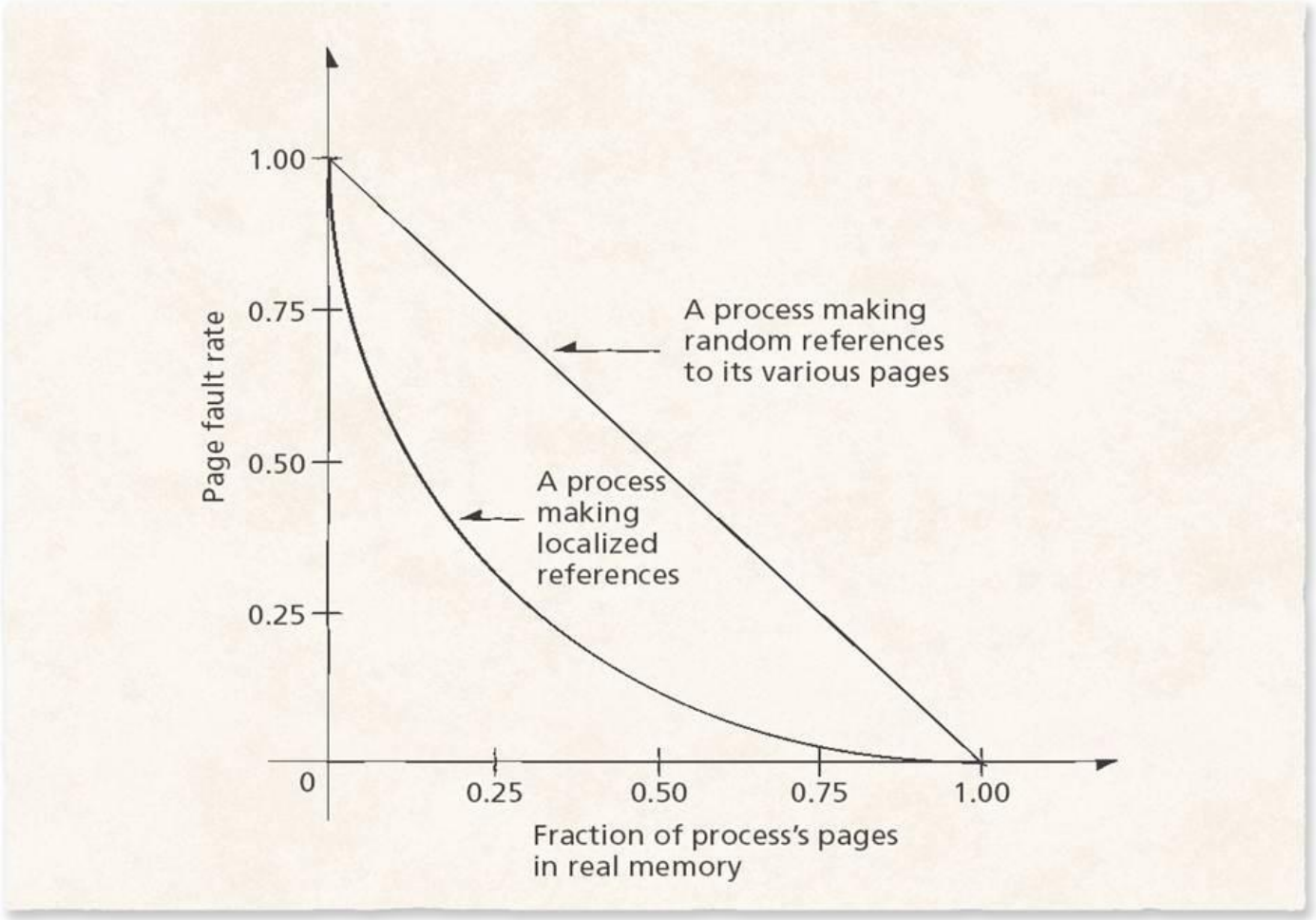
Working Set Model

- For a program to run efficiently
  - The system must maintain that program's favored subset of pages in main memory
- Otherwise
  - The system might experience excessive paging activity causing low processor utilization called thrashing as the program repeatedly requests pages from secondary storage
- Heuristic:
  - consider locality of page references
  - keep "local" pages of process in memory

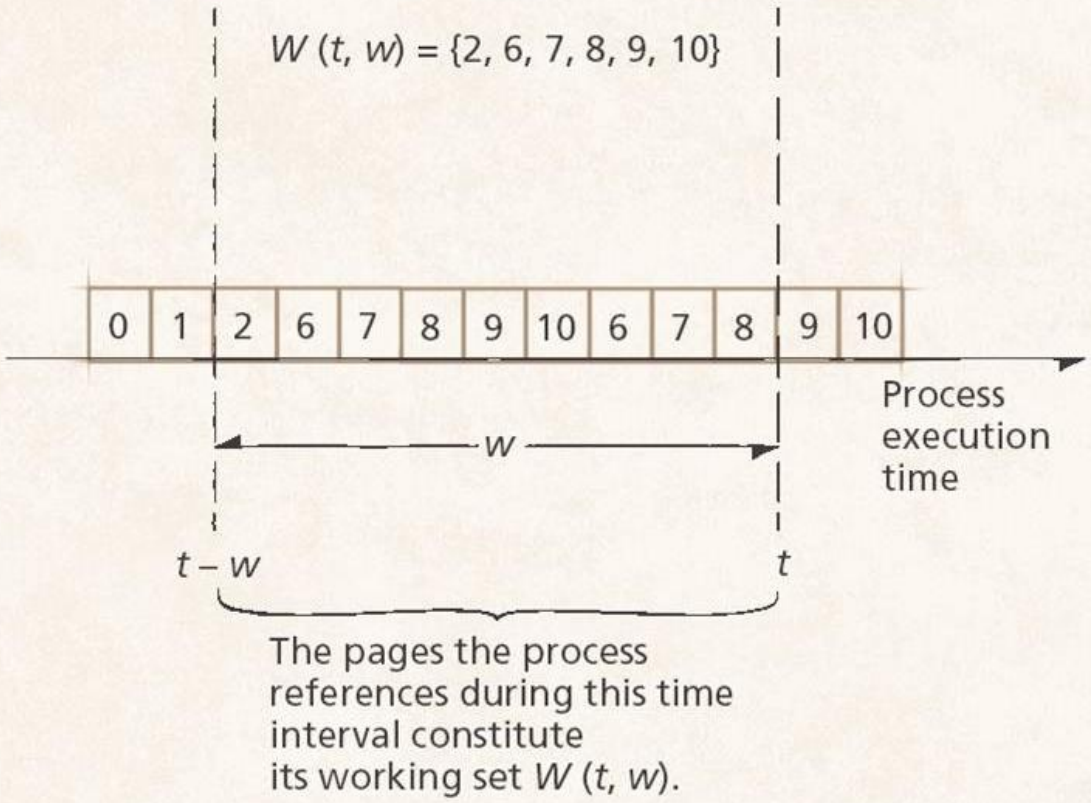
Example of page reference pattern



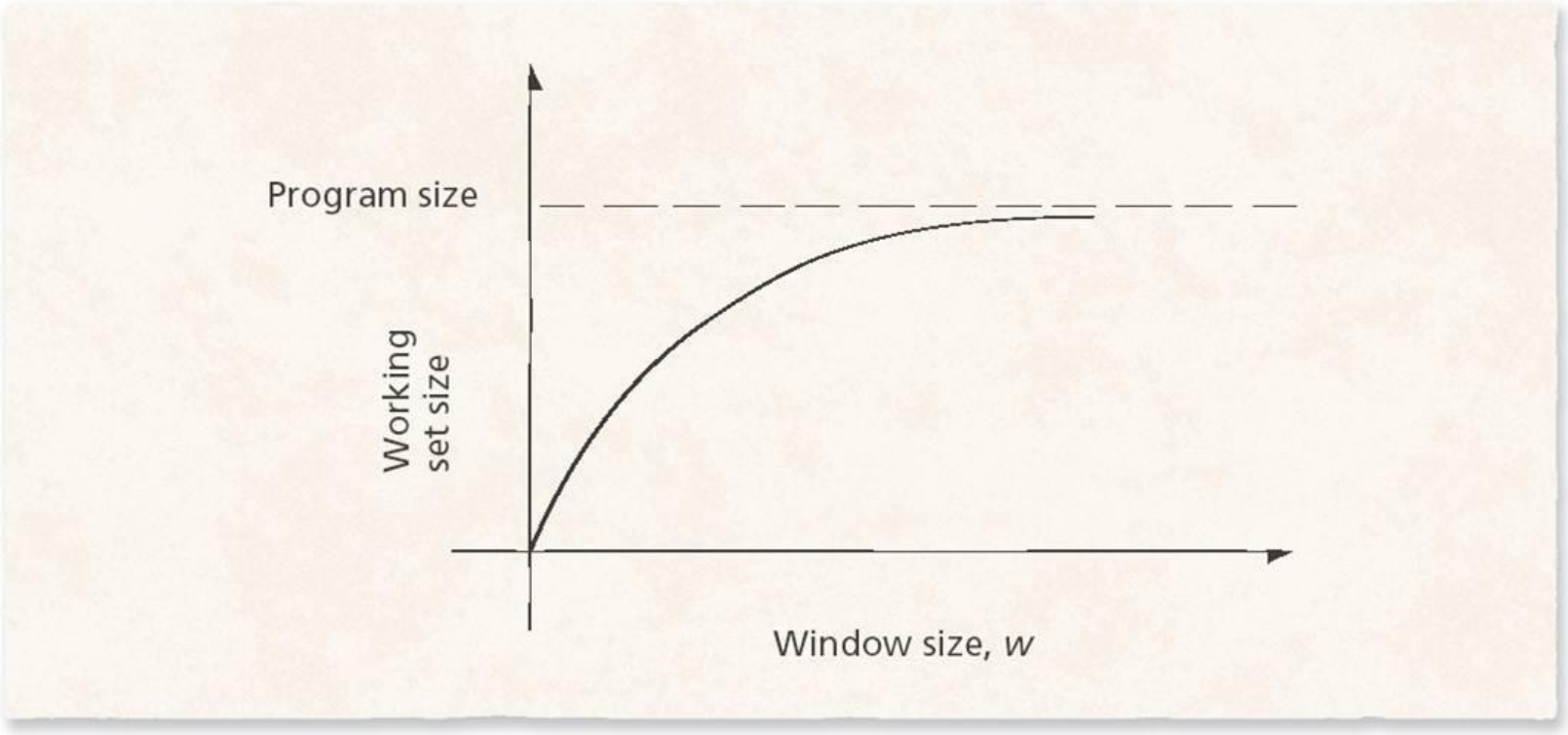
Effect of memory allocation to page fault



Concept: Working Set of process



Working Set Window size vs. program size



Working-Set-based page replacement strategy

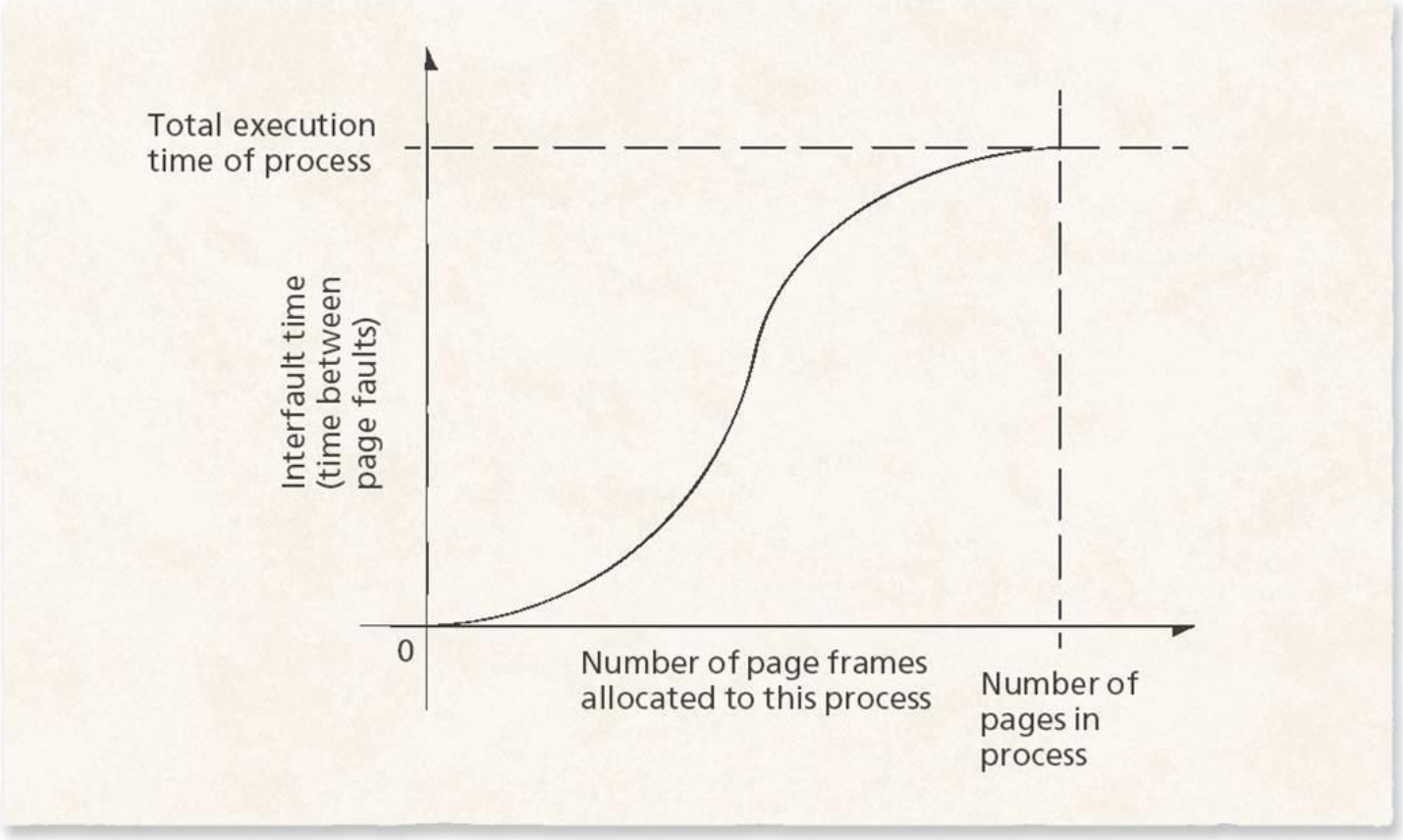
- keep pages of working set in main memory

- But:
  - working set size changes
  - working set changes
    - transition period yields ineffective memory use
  - overhead for working set management

Page-Fault-Frequency (PFF) Page Replacement

- Goal: improve working set approach
- Adjusts a process's resident page set
  - Based on frequency at which the process is faulting
  - Based on time between page faults, called the process's interfault time

Program Behavior under Paging



PFF Advantage

- Lower overhead
  - PFF (Page-Fault-Frequency)adjusts resident page set only after each page fault
  - Working set management must run after each memory reference

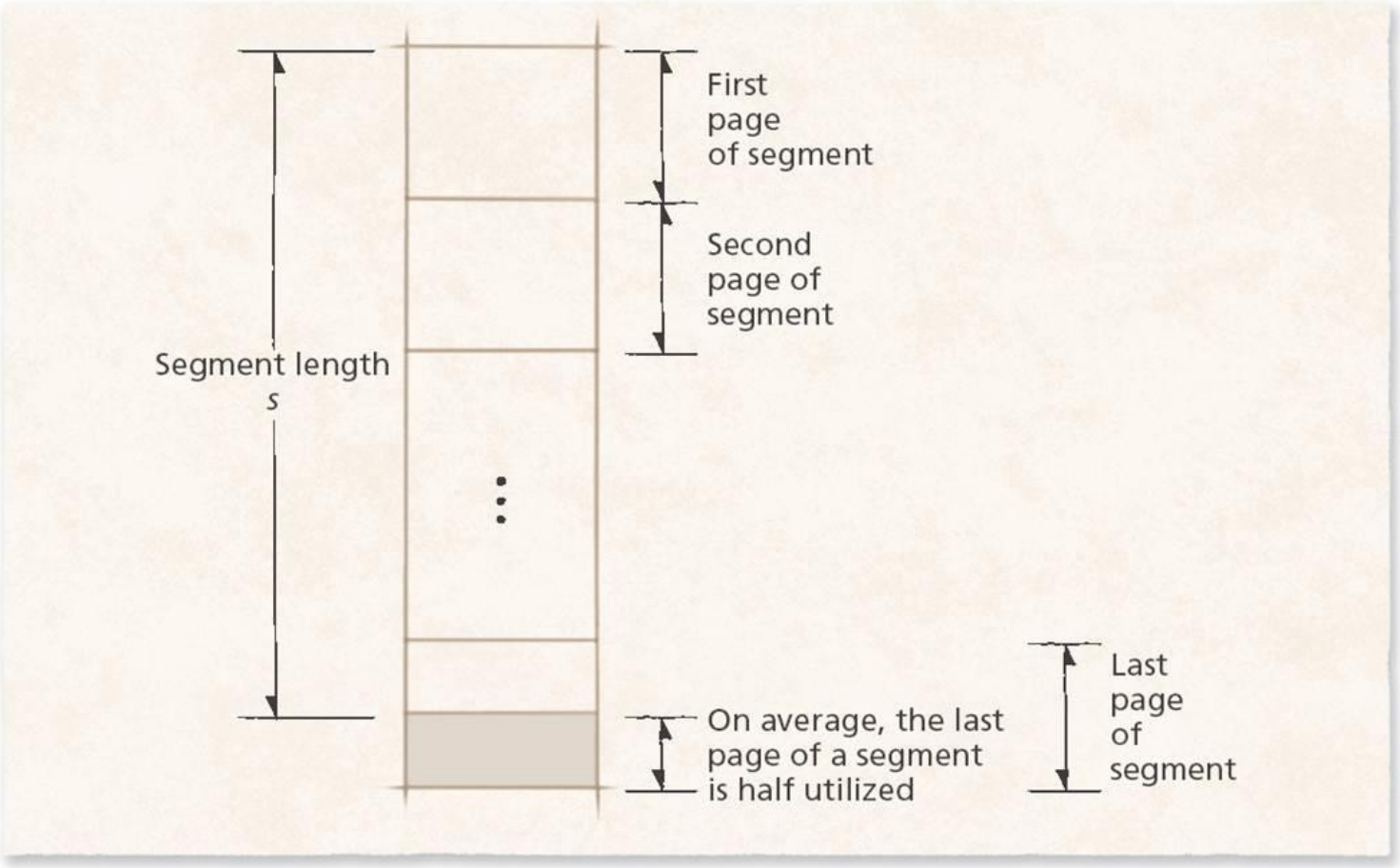
Page Release

- Problem: inactive pages may remain in main memory
- Solution:
  - explicit voluntary page release
  - need compiler and operating system support

Page Size

- Small page sizes
  - Reduce internal fragmentation
  - Can reduce the amount of memory required to contain a process's working set
  - More memory available to other processes
- Large page size
  - Reduce wasted memory from table fragmentation
  - Enable each TLB entry to map larger region of memory, improving performance
  - Reduce number of I/O operations the system performs to load a process's working set into memory

Page Size: internal fragmentation



Page Size examples

Manufacturer	Model	Page Size	Real address size
Honeywell	Multics	1KB	36 bits
IBM	370/168	4KB	32 bits
DEC	PDP-10 and PDP-20	512 bytes	36 bits
DEC	VAX 8800	512 bytes	32 bits
Intel	80386	4KB	32 bits
Intel / AMD	Pentium 4 / Athlon XP	4KB or 4MB	32- or 36 bits
Sun	UltraSparc II	8KB, 64KB, 512KB, 4MB	44 bits
AMD	Opteron / Athlon 64	4KB, 2MB and 4MB	32, 40, or 52 bits
Intel-HP	Itanium, Itanium 2	4KB, 8KB, 16KB, 64KB, 256KB, 1MB, 4MB, 16MB, 64MB, 256MB	Between 32 and 63 bits
IBM	PowerPC 970	4KB, 128KB, 256KB, 512KB, 1MB, 2MB, 4MB, 8MB, 16MB, 32MB, 64MB, 128MB, 256MB	32 or 64 bits

- Multiple page size
  - Possibility of external fragmentation

Global vs. Local Page Replacement

- Global: applied to all processes as a unit
  - ignore characteristics of individual process behavior
  - Global LRU (GLRU) page-replacement strategy
    - Replaces the least-recently-used page in entire system
    - Especially bad if used with RR scheduler
  - SEQ (sequence) global page-replacement strategy
    - Uses LRU strategy to replace pages until sequence of page faults to contiguous pages is detected, at which point it uses most-recently-used (MRU) page-replacement strategy
- Local: Consider each process individually
  - adjusts memory allocation according to relative importance of each process to improve performance